SUPPORT OF MAINSTAY SOFTWARE (USA) IN EUROPE

Database Manage

EMDAY International
71 rue des Atrebates
B-1040 Brussels - Belgium
Tel : 32-2-733.97.91
Tlx : 62239

Affranchir

S.V.P.

EMDAY International

Att : département vente par correspondance

34 Avenue de Tervueren bte 45
B-1040 Bruxelles - Belgique

Multi-user ISAM procedure set

The database set actually consists of four packages (with IDs 23, 24, 25 and 26). **MUISAM0.vep** contains procedures to set up the database parameter file and to convert this file into a standard text file. In **MUISAM1.vep** you will find the procedures to open and close the database, set the current data file and (indexed) record field, and to store or retrieve data from (VIP) record fields. **MUISAM2.vep** lets you manipulate the contents of the database, which is built on the **ISAM** (**I**ndexed **S**equential **A**ccess **M**ethod) model (C-Tree™ implementation by Faircom). **MUISAM3.vep** contains 3 utility procedures.
The packages use VIP records (class 11) and may be loaded independently, except for MUISAM2.vep which also needs MUISAM1.vep.
Like all VIP procedures, the database procedures update an internal status variable which, in case of error or failure, contains an error code accessible by the intrinsic function err().

MUISAM0.vep

set database params (*parameter file, result*)

parameter file (b): character string (constant or array)

result (B): simple object or array element of BYTE type

action: this procedure displays a dialog that lets you define the structure and parameters of a multi-user ISAM database. If the *parameter file* does not exist, it will be created.
The database may subsequently be opened and manipulated using the procedures in the packages MUISAM1.vep and MUISAM2.vep. The parameter file is only created or updated if the user clicks the **OK** button (*result* ≠ 0). If he clicks the **Cancel** button, the operation is cancelled and *result* will contain 0.
For each data file you must define:
- its name (any combination of letters, digits and '_')

For each record field you must define:
- its name (any combination of letters, digits and '_')
- if it is indexed or not
- if it allows duplicate or unique keys
- the type of data contained in the field

Once the database has been opened (*open database* procedure), you will refer to a data file or record field by its name.

param file to text (*parameter file*)

parameter file (b): character string (constant or array)

action: converts the given parameter file into a text file with the same name but ".txt" appended to it. Is useful for documentation.

get database (parameter file, record no)

parameter file (b): character string (constant or array)
record no (n): expression, simple object or array element of numeric type

action: fills the given record with an image of the database parameter structure contained in the file parameter file. The record's size is adjusted as needed. The record's structure will be as follows:

field	length in bytes
nb of data files:	1
for each data file:	
name length	1
name	variable
nb of fields	1
for each field:	
name length	1
name	variable
type	1
size	4
indexed	1
duplicates	1

"name" is a standard character string (any combination of letters, digits and '_') and "name length" includes the terminating null byte. The combined length of data file name and field name must be < 64 characters (for each field).

"type" = 0 text
1 integer number
2 date
3 picture
4 decimal number

"size" is only relevant if "type" = 0 and indicates the number of characters of a fixed-size text field (0 means variable size).
"indexed" = 1 or 0 for an indexed or unindexed field resp.
"duplicates" = 1 or 0 if an indexed field allows duplicates or not

set database (parameter file, record no)

parameter file (b):	character string (constant or array)
record no (n):	expression, simple object or array element of numeric type
action:	takes the image of a database parameter structure contained in the given record and builds the corr. parameter file bearing the given name. If the file already exists its previous contents will be overwritten. Extensive consistency checking is performed. Error codes are accessible via VIP's err() intrinsic function. Valid parameter structures should conform to the rules outlined above (see the *get database* procedure).

open database (*parameter file, directory*)

parameter file (b): character string (constant or array)
directory (b): character string (constant or array)

action: opens the database specified by its *parameter file* and located in the folder *directory* . Only one database may be open at any time. You must call this procedure before you can access the database. If the folder does not exist or if the database files cannot be found in the given folder, a new database will be created according to the specifications of the parameter file and will be located inside the given folder.

note: the current directory is saved before it is changed to *directory* , which is now the new current directory as long as the database remains open.

close database

action: closes the database that is currently open. YOU MUST CALL THIS PROCEDURE WHEN YOU ARE FINISHED WITH A DATABASE. An open database is associated with structures in memory which must be deleted when they are no longer needed.

note: the current directory is restored to what it was before you called *open database*

set data file (data file)

data file (b): character string (constant or array)

action: defines *data file* as the current data file for all subsequent operations.

set record field (record field)

record field (b): character string (constant or array)

action: defines *record field* as the current record field for all subsequent operations (assumes the field is indexed and belongs to the current data file, otherwise nothing happens).

put record field (*field name, record no, source*)

field name (b):	character string (constant or array)
record no (n):	expression, simple object or array element of numeric type
source (a):	object of any type (corresponding to the type of the field)
action:	fills a field of the VIP record *record no* using as template the record field *field name* of the database file *data file* . In the case of a character field, *source* should be a character string (constant or array). For integer and real fields, *source* is expected to be a simple object or array element of INTEGER or REAL type resp. For date fields a date array (as returned by the procedure *read date & time* (first 3 elements only)) is required. In the case of a picture, *source* should be the picture's identifier.
remark:	fixed length character fields are right-padded with blanks.

get record field (*field name, record no, destination*)

field name (b):	character string (constant or array)
record no (n):	expression, simple object or array element of numeric type
destination (B):	object of any type (corresponding to the type of the field)
action:	retrieves a field of the VIP record *record no* using as template the record field *field name* of the database file *data file* . In the case of a character field, *destination* should be a BYTE type array. For integer and real fields, *destination* is expected to be a simple object or array element of INTEGER or REAL type resp. For date fields an INTEGER array of at least 3 elements is required. If the field contains a picture, a copy is made of it and its identifier returned in *destination* (simple object or array element of BYTE type).

add record (*record no*)

record no (n):	expression, simple object or array element of numeric type
action:	adds the record identified by *record no* to the current data file.
remarks:	- the inserted record becomes the new current record for the current data file. - the corr. index file entries (keys) are also automatically inserted.

rewrite record (*record no*)

record no (n):	expression, simple object or array element of numeric type
action:	rewrites the current record for the current data file, according to the contents of the record *record no* .
remarks:	see *add record*

read record (*record no*)

record no (n):	expression, simple object or array element of numeric type
action:	reads the current data record relative to the current data file into the record *record no*.

delete record

action:	deletes the current record relative to the current data file.

first record (*record no, found*)

record no (n):	expression, simple object or array element of numeric type
found (B):	simple object or array element of BYTE type
action:	returns into the record *record no* the first data record in key sequential order relative to the current data file and record field. If the record is found, it becomes the new current record and *found* will be ≠ 0.

last record (*record no, found*)

record no (n): expression, simple object or array element of numeric type

found (B): simple object or array element of BYTE type

action: returns into the record *record no* the last data record in key sequential order relative to the current data file and record field. If the record is found, it becomes the new current record and *found* will be ≠ 0.

next record (*record no, found*)

record no (n): expression, simple object or array element of numeric type

found (B): simple object or array element of BYTE type

action: returns into the record *record no* the next data record in key sequential order relative to the current data file and record field. If the record is found, it becomes the new current record and *found* will be ≠ 0.

previous record (*record no, found*)

record no (n): expression, simple object or array element of numeric type

found (B): simple object or array element of BYTE type

action: returns into the record *record no* the previous data record in key sequential order relative to current data file and record field. If the record is found, it becomes the new current record and *found* will be ≠ 0.

search record (*key, record no, found*)

key (a):	simple object or array element of any type
record no (n):	expression, simple object or array element of numeric type
found (B):	simple object or array element of BYTE type
action:	searches the current data file for the first record whose current field is greater than or equal to the target key *key*. If such a record is found, it will be returned in the record *record no*. *Found* will contain one of the values: 0 record not found 1 record found with field = target key 2 record found with field > target key If the record has been found it becomes the new current record.

first record in set (*key, significic length, record no, found*)

key (a):	simple object or array element of any type
significic length (n):	expression, simple object or array element of numeric type
record no (n):	expression, simple object or array element of numeric type
found (B):	simple object or array element of BYTE type
action:	searches the current data file for the first record whose current field matches the given key *key* in the first*significic length* bytes. If such a record is found (*found* ≠ 0), it becomes the new current record and its contents are copied into the record *record no* . A successful search also defines a set of records matching the given criterion.
note:	this procedure cannot be used with real type fields

next record in set (*record no, found*)

record no (n): expression, simple object or array element of numeric type

found (B): simple object or array element of BYTE type

action: searches the record set defined by a previous and successful *first record in set* procedure for the next record in the set. If such a record is found, (*found* ≠ 0), it becomes the new current record and its contents are copied into the record *record no* .

lock database (*mode*)

mode (n): expression, simple object or array element of numeric type

action: sets the database locking mode according to the value of *mode* :

- 0 frees all data record locks held by your program and cancels the lock acquisition mode
- 1 resets acquisition mode = 0 + 2
- 2 enables acquisition mode: your program will acquire a lock on each record read
- 3 suspends acquisition mode: no locks are freed but no new locks will be acquired
- 4 restores acquisition mode: locks will be acquired again

note: this procedure is only useful in an environment where multiple updaters compete which each other; in this case you should acquire locks for the whole duration of your updates to ensure consistency.

Remarks : -all procedures which return data in VIP records will automatically adjust the record size if needed to contain the full data length
-key (indexed fields) comparisons are done according to the field type: (fixed length) character fields are scanned from left to right on a case insensitive basis, while integer, real and date fields are compared according to their numerical value.

MUISAM3.vep

WARNING: the utility procedures in this package should not be shared by multiple users. They ought to be reserved for persons with privileged access rights (e.g. database administrator).

rebuild database (*parameter file, directory*)

parameter file (b): character string (constant or array)
directory (b): character string (constant or array)

action: rebuilds the database specified by its parameter file and located inside the given directory. You will need to call this procedure if you have changed the database parameters (add or delete indexes e.g.) or otherwise if the database has become corrupt.

compact ISAM file (*ISAM file*)

ISAM file (b): character string (constant or array)

action: creates a new, compacted copy of an ISAM data or index file (deleted space removed). ".old" is appended to the name of the old (uncompacted) version of the file. The compacted file will have to be rebuilt (*rebuild database*) before it can be used again.

note: this procedure should only be called in rare occasions (with variable length record data files e.g.)

flat file to ISAM (*data file, record length*)

data file (b): character string (constant or array)
record length (n): expression, simple object or array element of numeric type

action: converts the fixed record length flat data file *data file* into an ISAM data file with the same name, ".flat" being appended to the flat file's name. All records of the flat file share the same *record length* and the first record begins with the first byte of the file.

ISAM error codes

The ISAM procedures may return the following error codes, to which a base value must be added, depending on the package:

ID = 23 --> base = 5888
ID = 24 --> base = 6148
ID = 25 --> base = 6403
ID = 26 --> base = 6656

2 key value already exists
3 could not delete since record positions don't match
4 could not find key to delete
5 cannot call delete w/o verification with duplicate keys
10 initialization parameters require too much space
11 bad initialization parameters
12 could not open index file
13 unknown file type
14 file corrupt at open
15 file has been compacted
16 could not create index file
17 could not create data file
18 tried to create existing index file
19 tried to create existing data file
20 key length too large for node size
21 record length too small
22 file number out of range
23 illegal index member info
24 could not close file
26 file number not active
28 zero position while adding key
29 zero position in data file routine
30 file pointer exceeds logical end of file
31 flag not set on record in delete chain
32 attempt to delete record twice in a row
33 attempt to use NULL pointer in read/write
34 predecessor repeat attempts exhausted
35 seek error (possible cause: out of disk space)
36 read error
37 write error (possible cause: out of disk space)
38 could not convert virtual open to actual
39 no more records available
40 key size too large for buffers
41 could not unlock data record
42 could not obtain data record lock
43 index file - version incompatibility

45 key length exceeds 64 bytes
46 file number already in use
48 file mode incompatibility
49 could not save file
50 could not lock node
51 could not unlock node
52 variable length keys disabled
100 no current record for data file
101 could not satisfy search request for index file
102 could not open ISAM parameter file
103 could not read first 5 parameters in ISAM parameter file
104 too many files in ISAM parameter file
106 could not read data file record in ISAM parameter file
107 too many keys for data file in ISAM parameter file
108 incorrect keyno for index member in ISAM parameter file
109 too many key segments defined in ISAM parameter file
110 could not read segment record in ISAM parameter file
111 could not read index file record in ISAM parameter file
112 lock protocol found pending locks
113 no space left in internal lock table
114 1st byte of data record equals delete flag (hex FF)
115 key segments do not match key length
117 could not read index member record
118 NXTSET called before FRSSET for index file
119 FRSSET called for index with wrong key type (decimal number)
120 data record length exceeds rebuild maximum
121 not enough space for sort area
122 attempt to change fixed vs variable length
123 variable length header has bad record mark
124 number of indices does not match
125 c-tree already initialized
140 variable record length exceeds 65,529 bytes
146 could not update free space information
147 space to be reused is not marked deleted
148 WRTVREC cannot fit record at file position
149 varlen less than minimum in ADDVREC
153 buffer too small in REDVREC
154 zero length record in REDVREC
158 REDVREC record not marked active
159 zero position in var record length function
160 multi-user interference: index information updated
by the time user got to actual data record

VipToMPW_C

Utilisation du MPW C

- créez dans le dossier VIP de votre disque dur, un dossier "VipToMPW" qui va contenir votre MPW complet (voir doc MPW)

- copiez de la disquette master le fichier "CompileVip.c" et placer le dans le dossier "Tools" du dossier MPW.

- copier le dossier "VipToMPW_C" en entier dans le dossier MPW.

!!! Attention ne pas changer le nom de ce dossier, ni les noms des fichiers qu'il contient. !!!

- après cette suite d'instruction, vous êtes prêt à travailler en MPW

- pour compiler un programme en MPW C, il suffit de translater le code VIP en code C MPW via le translateur. Il faut sauver le code C (après translation) dans le dossier VipToMPW_C. Notez qu'un programme (par exemple "Pipeline") doit être sauvé sous "Pipeline.c" par le translateur. Le ".c" est indispensable pour le compilateur MPW.

- pour effectuer la compilation et l'édition des liens, il faut entrer dans le MWP Shell en double cliquant dessus, puis taper la commande

<CompileVip.c> suivi du nom du fichier C a compiler.

Par Exemple : pour compiler "Pipeline" sauvez le dans le dossier "VipToMPW_C" sous le nom "Pipeline.c" et tapez la commande :

CompileVip.c "Pipeline" le ".c" ne doit pas être précisé, l'application sera sauvée sous le nom "Pipeline"

VipToMPW_PASCAL

Utilisation du MPW Pascal

- créez dans le dossier VIP de votre disque dur, un dossier "VipToMPW" qui va contenir votre MPW complet (voir doc MPW)

- copiez de la disquette master le fichier "CompileVip.p" et placer le dans le dossier "Tools" du dossier MPW.

- copier le dossier "VipToMPW_PASCAL" en entier dans le dossier MPW.

!!! Attention ne pas changer le nom de ce dossier, ni les noms des fichiers qu'il contient. !!!

- après cette suite d'instruction, vous êtes prêt à travailler en MPW

- pour compiler un programme en MPW Pascal, il suffit de translater le code VIP en code Pascal MPW via le translateur. Il faut sauver le code Pascal (après translation) dans le dossier VipToMPW_Pascal. Notez qu'un programme (par exemple "Pipeline") doit être sauvé sous "Pipeline.p" par le translateur. Le ".p" est indispensable pour le compilateur MPW.

- pour effectuer la compilation et l'édition des liens, il faut entrer dans le MWP Shell en double cliquant dessus, puis taper la commande

<CompileVip.p> suivi du nom du fichier Pascal a compiler.

Par Exemple : pour compiler "Pipeline" sauvez le dans le dossier "VipToMPW_Pascal" sous le nom "Pipeline.p" et tapez la commande :

CompileVip.p "Pipeline" le ".p" ne doit pas être précisé, l'application sera sauvée sous le nom "Pipeline"

SUPPORT OF MAINSTAY SOFTWARE (USA) IN EUROPE

With our compliments
Avec nos compliments

EMDAY International

71, rue des Atrebates • B-1040 Brussels • Belgium • Tel : 32-2-733.97.91 • Tlx : 62239